Assignment 3 Report

1046

The size of the grid and the amount of the numbers to add are read in. The amount of memory to store the table is then assigned and the numbers in the grid read in. To compare the adjacent numbers I realised that you only needed to compare four of them in relation to the pointers location 'x', and they are: to the right of 'x', down and left of 'x', straight below 'x' and, down and right of 'x' and each is compared 'm' times. This was needed as doing anymore comparisons would mean comparing numbers already compared and would be inefficient. Each of the four comparisons is kept inside the grid with 'if' statements that stop the program accessing memory outside the array. The value found in these methods is then compared to the current greatest and if it is larger, then the greatest is assigned to the new higher value.

1048

I first created a structure called 'date' to store the date information. I also created an array called 'months' that stored each month of the year. This was because it meant I could assign each month an integer value and compare these, instead of trying to compare strings, making the problem easier to solve. My function sortDates is called by the qsort and sorts the elements by comparing the years first, and if these are the same, then comparing the months integer value, and then if these are the same comparing the days. If the entire date is the same then they aren't reordered. In the main method the amount of dates to be entered is scanned in first. After this, that many date structs are read into an array, plus the date that the user will search for. The month that is read in is assigned as a temp value and then the integer value for the structure is assigned. The binary search is called by assigning the value to variable item. If item is found then the system will output yes, if it isn't then it will output no. to complete this task I used:
http://www.tutorialspoint.com/c_standard_library/c_function_qsort.htm to help with the qsort, and http://www.tutorialspoint.com/c_standard_library/c_function_bsearch.htm to help with the binary search.

1049

This task was accomplished by creating a struct to store the item and priority as this made it simpler than using an array to store them. A while loop is then used to read the first character in to determine what to do next. If it is an 'I' then we are inserting into the queue. This is done by seeing if the queue is empty, and if it is, inserting straight away. If it isn't then we compare the priority of the item to be inserted and the queue items to find its position. If it's position is at the end, then it is inserted, but if it is before the end, memmove is used to shift the queue down by one to allow space for the new item. If we are popping then we are seeing if the queue is empty, if it is then print '-1'. If it isn't then print the value at the top of the queue and then shift all items in the queue up by 1, and also decrease the queue length. Once the end of the file is reached we leave the while loop and the memory assigned to the queue is freed. To complete this task I used:
http://www.tutorialspoint.com/c_standard_library/c_function_realloc.htm to help me understand reallocating memory and
http://www.tutorialspoint.com/c_standard_library/c_function_memmove.htm to help with the moving of memory.

1050

Although I did not accomplish this task I would have set up a linked list to allow airports to be linked through one another. A type of search, probably A*, would have been used to search through the list to see if there was a connection between airports.